



# The 6 **levels** of Claude Code

A practical guide to how deep you're actually using Claude Code — what defines each level, the exact setup that gets you there, the trap waiting at each stage, and what's on the other side. Most people over-estimate by two levels.

## WHO THIS GUIDE IS FOR

|                   |                                                                                                                                                                          |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Beginners         | Only ever opened Claude Code and started typing? This shows exactly what you're missing, in order, nothing assumed.                                                      |
| Technical readers | Already running CLAUDE.md and a few skills? This shows precisely what separates you from parallel execution and live MCP connections — and why skipping ahead backfires. |

! **Be honest before you read on.** Most people are stuck on the first two levels and don't know it — they've built a slightly faster way to produce the same generic output an AI defaults to.



## 01 · LEVEL 1

# The chatbot **crutch**

A direct line from your prompt to Claude's output — no setup, no persistent context, no plan. It's a command-in, code-out machine, functionally no different from a chat window that happens to return runnable answers.

**WHAT IT LOOKS LIKE IN PRACTICE**

> Add a login screen to my app

- a No stack, no auth pattern, no design language.** Claude invents the framework, the fields, the error handling — every decision is a guess.
- b Guessing means regressing to the mean.** An LLM defaults to the statistical average of its training data — the most common way a login screen gets built across millions of repos. That average is generic by construction.
- c Inconsistency compounds.** Dozens of small, mutually inconsistent guesses — naming drifts, error handling drifts, structure drifts. Six weeks in, no two parts look built by the same process.

**Real skills you build here**

Writing clear, specific prompts. Reading output critically instead of trusting it because it compiles. Basic terminal literacy.

**How you know you're stuck**

Every prompt re-explains things you already told Claude in a previous session — framework, folder structure, conventions.



## 02 · LEVEL 2

# The partner

You reach Level 2 the moment you use Plan Mode before building — discussing the approach with Claude Code before a single line of code gets written, every time, not just for the hard problems.

- 1 Plan Mode + extended thinking.** Ask Claude to lay out how it would build it — files, approach, assumptions — and read that plan before approving. Surfaces bad assumptions while they're still a sentence, not a six-file diff.
- 2 Partner, not vending machine.** A design conversation, not a command. "Why this approach and not X?" "Compare two options before picking one."
- 3 Let Claude push back.** The step people skip and the one that matters most. An assistant that only agrees with your first idea isn't helping you think — it's just typing faster.

**COPY-PASTE · PLAN-FIRST PROMPT**

```
> I want to add a login screen. Before you write any code, tell me  
> your plan: what approach you'd take, which files you'll touch, and  
> flag anything you're assuming about my stack that you're not sure  
> about. Push back if you think there's a better way.
```

- ! The trap:** Plan Mode makes any single conversation better-reasoned. It gives Claude nothing that persists to the next session. You'll have the same "here's my stack" conversation tomorrow — words in a chat window are not memory.



## 03 · LEVEL 3

# Full context

The biggest single jump in the guide. You write and actively maintain a lean CLAUDE.md — the first file Claude Code reads every session, automatically. A prompt is heard once; CLAUDE.md is heard on every turn, for the life of the project.

## COPY-PASTE · MINIMAL CLAUDE.MD

```
# CLAUDE.md

## Stack
React Native (Expo, TypeScript). Firebase Auth + Firestore.
RevenueCat for subscriptions. No other backend.

## Folder structure
- /app          - screens (Expo Router, file-based)
- /components   - shared UI only, no screen-specific logic
- /hooks        - all Firestore/Firebase access lives here, nowhere else
- /lib          - pure utility functions, no side effects

## Rules
- Never call Firestore directly from a component. Always go through /hooks.
- Never add a new npm dependency without asking me first.
- All user-facing strings go through /lib/copy.ts - no inline strings.
- Design language: primary #2F7D5B, Inter, 8px spacing grid, 12px radius.

## Before you finish any task
- Run `npm run typecheck` and `npm run lint`. Both must pass.
- If you touched a hook, confirm the listener unsubscribes on unmount.
```

**Scaffold, then prune hard**

Run `/init` for a rough draft, then cut it down. Ask Claude to flag vague rules — you decide what each should say. Keep it under ~100 lines.

**`/clear` between tasks**

Finish a task, commit, `/clear` before the next unrelated one. A fresh context with a good CLAUDE.md beats a long, cluttered session.

! **The trap: a bloated CLAUDE.md.** Density over completeness. A wall of instructions dilutes the rules that matter now. Kept-wholesale `/init` output has repeatedly made agents perform worse — write and prune the final version yourself.



## 04 · LEVEL 4

# Skills and workflows

CLAUDE.md solves general context. Skills and workflows solve the specific, recurring tasks that would bloat it. This is where Claude Code starts to feel less like an assistant you brief and more like a team you've trained.

**Skills — scoped playbooks**

A folder with a detailed procedure for one recurring job. Loaded only when relevant, so it can be as detailed as the task needs without taxing every other request.

**Workflows — custom slash commands**

`/new-screen ProfileSettings` expands into a five-paragraph prompt. The thinking work happened once, when you wrote it.

**COPY-PASTE · SKILL EXAMPLE**

# Skill: Add a new authenticated data hook

Use this whenever adding a hook that reads/writes Firestore data scoped to the signed-in user.

**## Steps**

1. Create the hook in `/hooks/use.ts`
2. Read the current uid from ``useAuth()`` — never any other way
3. Scope the query with ``where('userId', '==', uid)`` at the query level — never filter client-side after an unscoped read
4. Return shape must be `{ data, loading, error }` — no exceptions
5. If it subscribes to a listener, confirm it unsubscribes in cleanup
6. Add a matching Firestore security rule before it's done

**## Example**

[a real, working hook from this exact codebase — concrete code]

! **The trap: too many skills.** 20–30 sharply-scoped skills beat 100 generic ones — a bad selection step is its own source of error. Gut check: *have I done this exact task 3+ times, wanting it repeated identically?*



## 05 · LEVEL 5

# Parallel agents and **execution**

You stop being the sole driver of one conversation and become closer to a lead directing several workstreams — multiple agents at once, each on its own Git branch, reviewed and merged independently.

## WHAT IT LOOKS LIKE OPERATIONALLY

```
# pane 1 – bug fix, own branch
git checkout -b fix/streak-calculation
> Fix useHabitStreak where a skipped day resets the streak to 0
> instead of the grace-period behavior in PRD.md.

# pane 2 – new feature, own branch
git checkout -b feature/widget-support
> Add the iOS home screen widget from TASKS.md, following the
> widget-support skill.

# pane 3 – tests, own branch
git checkout -b test/paywall-flow
> Write emulator tests for the paywall flow following the
> firestore-rules-testing skill. Cover success and denial cases.
```

- 1 Split along genuine boundaries** — agents that don't touch the same files. Two agents refactoring the same auth flow will collide.
- 2 Review each branch as it finishes**, merge to a shared branch frequently, and re-run the "before you finish" checklist on every branch. Tools like **Warp** or **Superset** coordinate this past a handful of agents.
- ! The trap: skipping the foundation.** Parallel agents amplify whatever foundation you give them. Without a lean CLAUDE.md and scoped skills you get 3× the slop, in 3 branches, each guessing differently — and the merge becomes a bigger project than the tasks.



## 06 · LEVEL 6

# Connecting with the outside world

Claude Code stops being confined to your local codebase and can check its assumptions against the real, live state of your product — through MCPs (Model Context Protocol servers) queried during a task.

**POSTHOG MCP · DEBUGGING A DROP-OFF**

```
> Look at the onboarding funnel in PostHog for the last 14 days.  
> Where's the biggest drop-off step, and has it moved compared to  
> the 14 days before that?
```

**SUPABASE MCP · CHECKING FOR SCHEMA DRIFT**

```
> Check the actual shape of documents in the habits collection  
> right now. Does it still match the schema described in CLAUDE.md,  
> or has it drifted?
```

- a Different in kind, not degree.** Every earlier level gave Claude better *static* context — which goes stale the moment your data or product state changes and nobody updates the docs.
- b The senior-engineer shift.** Not smarter reasoning in the abstract — just the habit of checking actual system state instead of a mental model that may no longer be accurate.



07 · RECAP

# The 6 levels at a glance

Claude Code's ceiling isn't the model — it's how much real context, structure, and connection you give it, and in what order you build them.

| LEVEL                  | CORE MECHANISM                                                           | THE TRAP                                                   |
|------------------------|--------------------------------------------------------------------------|------------------------------------------------------------|
| 1 · Chatbot Crutch     | Direct prompt → output, no context                                       | Claude guesses everything → slop                           |
| 2 · The Partner        | Plan Mode + extended thinking, inviting pushback                         | Words alone aren't persistent context                      |
| 3 · Full Context       | A hand-written, lean CLAUDE.md; <code>/clear</code> between tasks        | A bloated or unpruned AI-generated file makes agents worse |
| 4 · Skills & Workflows | Folder-based playbooks with real examples + slash commands               | Too many skills → Claude can't pick the right one          |
| 5 · Parallel Agents    | Parallel sessions on isolated branches, orchestrated via Warp / Superset | Skipping Levels 3–4 → parallel slop, expensive merges      |
| 6 · Outside World      | MCPs (PostHog, Supabase, ...) queried live during a task                 | —                                                          |

! **The biggest jump isn't the flashy one.** Not Level 5's parallel agents or Level 6's live connections — it's Level 2 → 3: the moment you stop re-explaining your project every session and give Claude something that persists. Skip it and everything above just makes your guesses faster and more numerous, not better.